Enriching Ontology-based Data Access with Provenance

Diego Calvanese\textsuperscript{1}, Davide Lanti\textsuperscript{1}, Ana Ozaki\textsuperscript{1}, Rafael Peñaloza\textsuperscript{2} and Guohui Xiao\textsuperscript{1}

\textsuperscript{1}KRDB Research Centre, Free University of Bozen-Bolzano, Italy
\textsuperscript{2}University of Milano-Bicocca, Italy

Abstract

Ontology-based data access (OBDA) is a popular paradigm for querying heterogeneous data sources by connecting them through mappings to an ontology. In OBDA, it is often difficult to reconstruct why a tuple occurs in the answer of a query. We address this challenge by enriching OBDA with provenance semirings, taking inspiration from database theory. In particular, we investigate the problems of (i) deciding whether a provenance annotated OBDA instance entails a provenance annotated conjunctive query, and (ii) computing a polynomial representing the provenance of a query entailed by a provenance annotated OBDA instance. Differently from pure databases, in our case these polynomials may be infinite. To regain finiteness, we consider idempotent semirings, and study the complexity in the case of DL-Lite\textsubscript{R} ontologies. We implement Task (ii) in a state-of-the-art OBDA system and show the practical feasibility of the approach through an extensive evaluation against two popular benchmarks.

1 Introduction

Ontology-based data access (OBDA) [Xiao et al., 2018] is by now a popular paradigm which has been developed in recent years to overcome the difficulties in accessing and integrating legacy data sources. In OBDA, users are provided with a high-level conceptual view of the data in the form of an ontology that encodes relevant domain knowledge. The concepts and roles of the ontology are associated via declarative mappings to SQL queries over the underlying relational data sources. Hence, user queries formulated over the ontology can be automatically rewritten, taking into account both ontology axioms and mappings, into SQL queries over the sources.

When issuing a query, in many settings it is crucial to know not only its result but also how it was produced, how many different ways there are to derive it, or how dependent it is on certain parts of the data [Senellart, 2017; Zimmermann et al., 2012; ?]. To address these issues, which are of importance already for plain relational database management systems (RDBMSs), \textit{provenance semirings} [Green et al., 2007; Green and Tannen, 2017] were introduced as an abstract tool to record and track provenance information; that is, to keep track of the specific database tuples that are responsible for deriving an answer tuple, and of additional information associated to them. In OBDA, determining provenance is made even more challenging by the fact that answers are affected by implicit consequences derived through ontology axioms, and by the use of mappings. Such elements come indirectly into play in query rewriting, hence provenance information must be reconstructed from the rewritten queries used in the answering process [Borgida et al., 2008].

In this work, we start from the semiring approach introduced for RDBMSs, and extend it to the full-fledged OBDA setting. To do so, we assume that not only database tuples are annotated with a label representing provenance information (e.g., the data source or the relation in which the tuple is stored), but also mappings and ontology axioms. Then, our task is to derive which combinations of these labels lead to the answer of a query. Such information is expressed through a \textit{provenance polynomial}, as illustrated in the following example.

\textbf{Example 1.} Let \texttt{Mayors[Person, City]} be a database relation with the tuples (\texttt{Renier, Venice}) and (\texttt{Brugnaro, Venice}), annotated with (sources) \texttt{p} and \texttt{q}, respectively. Assume two mappings \texttt{City(Y)} $\leftarrow$ \texttt{Mayors(X,Y)} and \texttt{headGov(X,Y)} $\leftarrow$ \texttt{Mayors(X,Y)}, annotated with \texttt{m} and \texttt{n}, respectively. The mappings and the database induce (i) two times the DL assertion \texttt{City(Venice)}, one annotated with \texttt{p $\times$ m} and one with \texttt{q $\times$ m}, (ii) the DL assertion \texttt{headGov(Renier, Venice)}, annotated with \texttt{p $\times$ n}, and (iii) the assertion \texttt{headGov(Brugnaro, Venice)}, annotated with \texttt{q $\times$ n}.

Now consider the inclusion \texttt{$\exists$s : headGov $\sqsubseteq$ Mayor} annotated with \texttt{s}. The answer true to the Boolean conjunctive query \texttt{$\exists x$.(Mayor($x$))} can be derived using this inclusion and any of the last two DL assertions. This information can be expressed through the provenance polynomial \texttt{((p $\times$ n) + (q $\times$ n)) $\times$ s}.

In our OBDA setting, concept and role inclusions of the ontology affect query results, as illustrated in Example 1. By annotating the inclusions and the mappings, in addition to the tuples, we can distinguish which inclusions and mappings were involved in the derivation of a query result. This differs from the approach proposed for attributed DL-Lite\textsubscript{R} [Bourgaux and Ozaki, 2019], where the inclusions are used to express constraints on the provenance information.

We investigate the problems of (i) deciding whether a provenance annotated OBDA instance entails a provenance anno-
tated conjunctive query (CQ), and (ii) computing a provenance
of a CQ entailed by a provenance annotated
OBDA instance. Differently from plain databases, in our case
these polynomials may be infinite. To regain finiteness, we
consider idempotent semirings, and study the complexity for
DL-LiteR ontologies [Calvanese et al., 2007]. We implement
(iii) in the state-of-the-art OBDA system Ontop [Calvanese et al.,
2017], and show the practical feasibility of our approach
through a detailed evaluation against two popular benchmarks.

An extended version of this work is available as a technical
report [?].

2 Basic Definitions
We represent the provenance information via a positive algebra
provenance semiring (or provenance semiring for short), originally
introduced for databases [Green et al., 2007]. Given a
countably infinite set \( N_V \) of variables, the provenance semiring
is the algebra \( K = (\mathbb{N}[N_V], \times, 0, 1) \), where \( \mathbb{N}[N_V] \) denotes the
space of polynomials with coefficients in \( \mathbb{N} \) and variables
in \( N_V \), the product \( \times \) and the addition \( + \) are two commutative
and associative binary operations over \( \mathbb{N}[N_V] \), and \( \times \) distributes over \(+\). A monomial from \( K \) is a finite product
of variables in \( N_V \). \( N_M \) and \( N_P \) denote the sets of all monomials
from \( K \), and of all finite sums of monomials in \( N_M \), respectively; i.e., \( N_P \) contains only polynomials of the form
\( \sum_{i \leq m} \prod_{j \leq m_i} a_{i,j,i} \), with \( a_{i,j,i} \in N_V \), and \( n, m_i > 0 \).
Since all coefficients are in \( \mathbb{N} \), they disappear in this expanded
form; e.g., \( 2a \) is \( a + a \). A polynomial in expanded form is a
finite sum of monomials, each formed by a finite product of
variables. By distributivity, every polynomial can be equiva-
lently rewritten in expanded form; however, the expanded
form of a polynomial may become exponentially larger. By
our definitions, \( N_V \subseteq N_M \subseteq N_P \).

2.1 Annotated OBDA
The provenance information of each axiom in an ontology,
each mapping, and each tuple in a data source, is stored as an
annotation. For this paper, we consider the standard OBDA
setting with ontologies written in DL-LiteR [Calvanese et al.,
2007], standard relational databases as data sources, and
mappings given by GAV rules. Consider three mutually disjoint
countable sets of concept names \( N_C \), role names \( N_R \), and
individual names \( N_I \). Assume that these sets are also disjoint from
\( N_V \), DL-LiteR role and concept inclusions are expressions of
the form \( S \sqsubseteq T \) and \( B \sqsubseteq C \), respectively, where \( S, T \) are role
expressions and \( B, C \) are concept expressions built through
the grammar rules

\[
S ::= R \mid R^\star, T ::= S \mid \neg S, B ::= A \mid \exists S, C ::= B \mid \neg B,
\]

with \( R \in N_R \) and \( A \in N_C \). A DL-LiteR axiom is a DL-LiteR
role or concept inclusion. An annotated DL-LiteR ontology is a finite set of annotated axioms of the form \( (\alpha, p) \), where \( \alpha \) is a DL-LiteR axiom and \( p \in N_R \).

A schema \( S \) is a finite set of predicate symbols disjoint from
\( N_C \cup N_R \) with \( \text{ar}(P) \) the arity of \( P \in S \). An annotated
data instance \( D \) over \( S \) maps every \( P \in S \) to a finite subset
\( P^D \) of \( N_M^{|\text{ar}(P)} \times N_V \). An annotated mapping is a finite set of
annotated rules \((\rho, p)\), where \( \rho \) is a (GAV) rule and \( p \in N_V \).

A rule \( \rho \) is of the form \( E(\vec{x}) \leftarrow \varphi(\vec{x}, \vec{y}, \vec{z}), \) with \( E \in N_C \cup N_R \) and
\( \varphi(\vec{x}, \vec{y}, \vec{z}) \) a conjunction of atoms \( P(\vec{t}, \vec{i}), \) with \( P \in S, \vec{t}, \vec{i} \) an \( \text{ar}(P) \)-tuple of terms in \( \vec{x} \cup \vec{y} \), and \( t \in \vec{t} \). We restrict \( \varphi \) to a conjunction of atoms for simplicity of our theoretical development,
also in line with the idea that semirings capture the
provenance of positive queries [Green et al., 2007]. See Sec. 5
for handling arbitrary OBDA mappings in our implementation.

An annotated OBDA specification \( \mathcal{P} \) is a triple \((\mathcal{O}, \mathcal{M}, S)\),
where \( \mathcal{O} \) is an ontology with annotated axioms, \( S \) is a data
source schema whose signature is disjoint from the signature of
\( \mathcal{O} \), and \( \mathcal{M} \) is a set of annotated mappings, connecting \( S \) to \( \mathcal{O} \) [Xiao et al., 2018]. The pair \((P, D)\) of an annotated
OBDA specification \( \mathcal{P} \) and an annotated data instance \( D \) is an
annotated OBDA instance. In OBDA, data sources and
mappings induce virtual assertions. In annotated OBDA,
virtual assertions are annotated with the provenance information
of the mapping and of matching tuples in the data instance.
Formally, an annotated assertion \((E(\vec{a}), p)\) is an expression of the form \((A(a), p) \) or \((R(a, b), p)\), with \( A \in N_C, R \in N_R, a, b \in N_V \), and \( p \in N_M \). We write \( \varphi(\mu(\vec{x}, \vec{y}, \vec{z})) \subseteq D \) if \( \mu \) is a
function mapping \( \vec{x}, \vec{y} \) to \( N_V \), \( z \) to \( N_V \), and \( (\mu(\vec{t}, \vec{i})) \in P^D \), for every atom \( P(\vec{t}, \vec{i}) \) in \( \varphi(\vec{x}, \vec{y}, \vec{z}) \). Given an annotated mapping
\( \mathcal{M} \) and data instance \( D \), the set \( \mathcal{M}(D) \) of annotated assertions

\[
(E(\mu(\vec{x})), p \times \prod_{z \in \vec{z}} h(z)), \text{ satisfying }
\]

\( (E(\vec{x}) \leftarrow \varphi(\vec{x}, \vec{y}, \vec{z}), p) \in \mathcal{M} \) and
\( \varphi(\mu(\vec{x}, \vec{y}, \vec{z})) \subseteq D \) is the set of virtual annotated assertions for \( \mathcal{M} \) over \( D \).

The semantics of annotated OBDA instances is based on
interpretations over the signature of the ontology, extending classical
DL-LiteR interpretations to track provenance, when relevant.
An annotated interpretation is a triple \( \mathcal{I} = (\Delta^I, \Delta^m^I, \Delta^r^I) \)
where \( \Delta^I \) and \( \Delta^m^I \) are non-empty disjoint sets (called the
domain of \( \mathcal{I} \) and the domain of monomials of \( \mathcal{I} \), respectively), and \( \Delta^r^I \) is the annotated interpretation function mapping

- every \( a \in N_I \) to some \( a^I \in \Delta^I \);
- every \( p, q \in N_M \) to some \( p^I, q^I \in \Delta^m^I \) s.t. \( p^I = q^I \) iff
  the monomials \( p \) and \( q \) are mathematically equal (modulo
  associativity and commutativity, e.g., \((p \times q)^I = (q \times p)^I \)
  by commutativity);
- every \( A \in N_C \) to some \( A^I \subseteq \Delta^I \times \Delta^m^I \); and
- every \( R \in N_R \) to some \( R^I \subseteq \Delta^I \times \Delta^I \times \Delta^m^I \).

We extend \( \Delta^I \) to further DL-LiteR expressions as natural:

\[
\begin{align*}
(R^-)^I &= \{(c, d, p^I) \mid (d, e, p^I) \in R^I\}, \\
(\neg S)^I &= \{\Delta^I \times \Delta^m^I \setminus S^I\}, \\
(\exists S)^I &= \{(d, p^I) \mid \exists e \in \Delta^I : (d, e, p^I) \in S^I\}, \text{ and} \\
(\neg B)^I &= \{\Delta^I \times \Delta^m^I \setminus B^I\}.
\end{align*}
\]

The annotated interpretation \( \mathcal{I} \) satisfies:

\[
\begin{align*}
(A(a), p) & \text{ in } \mathcal{I} \iff a^I \in A^I, \\
(R(a, b), p) & \text{ in } \mathcal{I} \iff a^I \times b^I \in R^I, \\
(B \subseteq C, p) & \text{ in } \mathcal{I} \iff \forall q \in N_M(q, d_q^I) \in B^I \implies (d, q \times p^I) \in C^I, \text{ and} \\
(S \subseteq T, p) & \text{ in } \mathcal{I} \iff \forall q \in N_M(q, d_q^I) \in S^I \implies (d, e, p^I) \in T^I.
\end{align*}
\]

\( \mathcal{I} \) satisfies an annotated ontology \( \mathcal{O} \), in symbols \( \mathcal{I} \models \mathcal{O} \), if it satisfies all annotated axioms in \( \mathcal{O} \). \( \mathcal{I} \) satisfies an annotated
OBDA instance \(((\mathcal{O}, \mathcal{M}, S), D)\) if \( \mathcal{I} \models \mathcal{O} \) and \( \mathcal{I} \models \mathcal{M}(D) \).
Example 2. Consider the OBDA instance of Example 1 and an annotated interpretation $I$ with $\Delta^I = \{\text{Renier, Venice, Brugnaro}\}$, $\Delta^I_{\text{prov}}$ containing $p \times n$, $q \times n \times m$, $p \times m$, $p \times n \times s$, $q \times n \times s$, with such individuals and monomials interpreted by themselves, and

\[
\begin{align*}
\text{headGov}^I &= \{(R, V, p \times n), (B, V, q \times n)\}, \\
\text{Mayor}^I &= \{(R, p \times n \times s), (B, q \times n \times s)\}, \\
\text{City}^I &= \{(V, p \times m), (V, q \times m)\}.
\end{align*}
\]

$I$ is a model of such OBDA instance, where R, V, and B stand for Renier, Venice, and Brugnaro, respectively. $
\qquad\square$

Following the database approach [Green et al., 2007; Green and Tannen, 2017], we annotate facts in interpretations with provenance information. However, in Green et al.’s setting, the database “is” the (only) interpretation, while in our case we adopt the open world assumption (as in OBDA), so the semantics is based on multiple interpretations. Our semantics ensures that, if we have a tuple $(d, p^2) \in C_I^2$ and $(C \subseteq D)$ is annotated with $n$, then $(d, (p \times n)^2) \in D^\geq$. So derivations are also represented in interpretations, and thus can be entailed. Each derivation is independent of the others.

Regarding the semantics of negation, we point out that, at the level of an interpretation, the lack of provenance information is a support for the negation of a fact. This apparent counterintuitive behaviour does not hold in all interpretations, hence it does not manifest in the entailments. In fact, our focus in this paper is query entailment (defined next), negations are only defined to comply with the usual syntax and semantics of DL-Lite$_R$. They do not affect query results, as in DL-Lite$_R$.

### 2.2 Annotated Queries

We extend the notion of conjunctive queries in DLs by allowing binary and ternary predicates, where the last term of a tuple may contain provenance information represented as a monomial (by definition of the semantics of annotated OBDA instances, the last element of a tuple can only contain monomials, not sums). More specifically, a **Boolean conjunctive query (BCQ)** $q$ is a sentence $\exists \vec{x}. \varphi(\vec{x}, \vec{a}, \vec{p})$, where $\varphi$ is a conjunction of (non-repeating) atoms of the form $A(t_1, t_2)$, $R(t_1, t_2, t_3)$, and $t_i$ is either an individual name from $\vec{a}$, or a variable from $\vec{x}$, and the last term of each tuple is either an element of $N_M$ in the list $\vec{p}$ or a variable from $\vec{x}$. We often write $P(\vec{t}, t)$ to refer to an atom which can be either $A(t_1, t_2)$ or $R(t_1, t_2, t)$ and $P(\vec{t}, t) \in q$ if $P(\vec{t}, t)$ is an atom occurring in $q$.

A **match** of the BCQ $q = \exists \vec{x}. \varphi(\vec{x}, \vec{a}, \vec{p})$ in the annotated interpretation $I$ is a function $\pi : \vec{x} \cup \vec{a} \cup \vec{p} \rightarrow \Delta^I \cup \Delta^I_{\text{prov}}$ such that $\pi(b) = b^\geq$, for all $b \in \vec{a} \cup \vec{p}$, and $\pi(t_1) \in P^I$, for every $P(\vec{t}, t) \in q$. $I$ satisfies the BCQ $q$, written $I \models q$, if there is a match of $\pi$ in $I$. A BCQ is entailed by an annotated OBDA instance if it is satisfied by every model of it. For a BCQ $q$ and an interpretation $I$, $\text{prov}_I^q(q)$ denotes the set of all matches of $q$ in $I$. The **provenance of $q$ on $I$**, denoted $\text{prov}_I^q(q)$, is the (potentially infinite) expression:

$$\sum_{\pi \in \text{prov}_I^q(q)} \prod_{P(\vec{t}, t) \in q} \pi^{-1}(t)$$

where $\pi(t)$ is the last element of the tuple $\pi(\vec{t}, t) \in P^I$; and $\pi^{-1}(t)$ is any $v \in N_M$ s.t. $v^2 = \pi(t)$. For $p \in \mathbb{N}_p$, we write $p \subseteq \text{prov}_I^q(q)$ if $p$ is a sum of monomials and for each occurrence of a monomial in $p$ we find an occurrence of it in $\text{prov}_I^q(q)$. $I$ satisfies $q$ with provenance $p \in \mathbb{N}_p$, written $I \models (q, p)$, if $I \models q$ and $p \subseteq \text{prov}_I^q(q)$.

The annotated OBDA instance $(\mathcal{P}, \mathcal{D})$ **entails** $q$, $(\mathcal{P}, \mathcal{D}) \models q$, if for all annotated interpretations $I$, if $I \models (\mathcal{P}, \mathcal{D})$ then $I \models q$; and $(\mathcal{P}, \mathcal{D}) \models (q, p)$, if $(\mathcal{P}, \mathcal{D}) \models q$ and $p \subseteq \text{prov}_I^q(q)$, for all $I$ satisfying $(\mathcal{P}, \mathcal{D})$.

In our syntax, the atoms of the queries contain an additional parameter which may either be a variable or a monomial. As a result, one can filter query results based on provenance information by specifying constraints in the last parameter of the atoms, which was not possible in the original approach by Green et al. [Green et al., 2007; Green and Tannen, 2017]. For example, $\exists xy. A(x, p) \land B(x, y)$ can be used to specify that we are only interested in matches of the query where the first atom is associated with a particular provenance. Variables can also be repeated, e.g. $\exists xy. A(x, y) \land B(x, y)$. One can fall back to the original setting from databases, where no constraints are imposed, by simply associating the last term of each atom with a fresh variable (see standard queries in Section 4).

The size $|X|$ of an annotated OBDA instance, a polynomial or a BCQ $X$ is the length of the string that represents $X$. We assume a binary encoding of elements of $N_C N_R N_d N_p$ occurring in $X$. We may omit ‘annotated’ in front of terms such as ‘OBDA,’ ‘queries,’ ‘inclusions,’ ‘assertions,’ and others, whenever this is clear from the context.

### 2.3 Reasoning Problems

Annotating OBDA instances with provenance information does not impact consistency checking. That is, an annotated OBDA instance is satisfiable precisely when the OBDA instance that results from removing the annotations is satisfiable. We thus focus on the problem of query entailment w.r.t. a provenance polynomial: given an (annotated) OBDA instance $(\mathcal{P}, \mathcal{D})$, a query $q$ and a polynomial $p \in \mathbb{N}_p$ decide if $(\mathcal{P}, \mathcal{D}) \models (q, p)$. Another important and related problem is to compute the provenance of a query: given an OBDA instance $(\mathcal{P}, \mathcal{D})$ and a query $q$, compute the set of all $p \in \mathbb{N}_p$ such that $(\mathcal{P}, \mathcal{D}) \models (q, p)$. In our formalism, the latter problem depends on whether there is a finite set of polynomials which we can compute. As shown next, in DL-Lite$_R$ the set of provenance polynomials may be infinite.

**Example 3.** Consider an OBDA instance $(\mathcal{P}, \mathcal{D})$ as in Ex. 1, but where now $O$ of $\mathcal{P}$ contains also $\text{Mayor} \subseteq \exists \text{headGov} \cdot t$. For all $i \in \mathbb{N}$, $(\mathcal{P}, \mathcal{D}) \models (\text{Mayor}(\text{Renier}), p \times n \times s^i + 1 \times t^i)$. Indeed, for any model $I$ of $(\mathcal{P}, \mathcal{D})$, $(\text{Renier}, (p \times n \times s^i + 1 \times t^i)) \in \text{Mayor}^I$ implies $(a, (p \times n \times s^i + 1 \times t^i)^2) \in (\exists \text{headGov})^I$, which implies $(\text{Renier}, (p \times n \times s^i + 1 \times t^i)^2) \in \text{Mayor}^I$, and so on. $
\square$

In Section 3 we consider the problem of query entailment w.r.t. a provenance polynomial. Note that in Example 3, if the semiring is multiplicatively idempotent (i.e., $s \times s = s$), the set of provenance polynomials is finite: the only polynomial is $p \times n \times s^i \times t$. This is not a coincidence; under multiplicatively-idempotency, the set of provenance polynomials is always finite. The following proposition states that multiplicative-idempotency is indeed sufficient to guarantee a finite set of polynomials.
Proposition 1. Under multiplicative idempotency, for any satisfiable OBDA instance \((\mathcal{P}, \mathcal{D})\) and BCQ \(q\), the set of polynomials \(p \in \mathbb{N}_p\) such that \((\mathcal{P}, \mathcal{D}) \models (q, p)\) is finite.

In Section 4 we study idempotent semirings and consider the problem of computing the provenance of a query.

### 3 Provenance Annotated Query Entailment

We establish complexity results for the problem of deciding whether an OBDA instance entails a (provenance annotated) query. For clarity of presentation, we split our proof in two parts. We first show that for an OBDA instance \((\mathcal{P}, \mathcal{D})\) and a query \((q, p)\), there is an OBDA instance \((\mathcal{P}_m, \mathcal{D}_m)\) and a set \(\text{Tr}(q_m, p_m)\) of (non-annotated) queries such that \((\mathcal{P}, \mathcal{D}) \models (q, p)\) if \((\mathcal{P}_m, \mathcal{D}_m)\) entails some \(q' \in \text{Tr}(q_m, p_m)\). Moreover, the sizes of \((\mathcal{P}_m, \mathcal{D}_m)\) and \(q'\) are polynomial in the sizes of \((\mathcal{P}, \mathcal{D})\) and \((q, p)\). Then, we adapt the query rewriting algorithm PerfectRef [Calvanese et al., 2007] to decide whether \((\mathcal{P}_m, \mathcal{D}_m) \models q'\).

#### 3.1 Characterization

Lemma 1 states that, given an OBDA instance \((\mathcal{P}, \mathcal{D})\) and a query \((q, p)\), there is an OBDA instance \((\mathcal{P}_m, \mathcal{D}_m)\) and a query \((q_m, p_m)\) that can be used to decide \((\mathcal{P}, \mathcal{D}) \models (q, p)\) and, moreover, all monomials in \(p_m\) are mathematically distinct (modulo associativity, commutativity, and distributivity).

**Lemma 1.** Given a satisfiable OBDA instance \((\mathcal{P}, \mathcal{D})\) and a query \((q, p)\), there are \((\mathcal{P}_m, \mathcal{D}_m)\) and \((q_m, p_m)\) such that

- any two monomials \(p_1, p_2\) appearing in \(p_m\) are mathematically distinct;
- \((\mathcal{P}, \mathcal{D}) \models (q, p)\) iff \((\mathcal{P}_m, \mathcal{D}_m) \models (q_m, p_m)\); and
- \(|\langle \mathcal{P}_m, \mathcal{D}_m \rangle| + |q_m, p_m\rangle\) is polynomially bounded by \(|\langle \mathcal{P}, \mathcal{D} \rangle| + |q, p\rangle|\).

We show that, given \((\mathcal{P}_m, \mathcal{D}_m)\) and \((q_m, p_m)\) as in Lemma 1, \((q_m, p_m)\) can be translated into a set of queries such that \((\mathcal{P}_m, \mathcal{D}_m)\) entails \((q_m, p_m)\) if it entails at least one of these queries. We first define the translation of a BCQ where all terms are variables (no individual names and no polynomials), and then adapt the translation for the general case. Given the BCQ \(q_m = \exists \vec{x}. \phi(\vec{x})\) with \(k\) atoms and \(p_m \in \mathbb{N}_p\) monomials, define \(\text{Tr}(q_m, p_m)\) as the set of all BCQs:

\[
\exists \vec{y}. \bigwedge_{1 \leq i \leq n} \phi_i(\vec{x}_i),
\]

where \(\vec{y} = \vec{x}_1, \ldots, \vec{x}_n\) and each \(\phi_i(\vec{x}_i)\) is a ‘copy’ of \(\phi(\vec{x})\) in which we replace each variable \(x \in \vec{x}\) by a fresh variable \(x_i \in \vec{x}_i\). We check whether we can find the monomials of the polynomial in these matches by replacing the last variable in each \(j\)-th atom of \(q_m\) by a monomial \(p_{i,j} \in \mathbb{N}_m\) built from symbols occurring in \(p_m\) such that \(\prod_{1 \leq i \leq k} p_{i,j} = p_i\) for some \(p_i \in \mathbb{N}_p\), with \(1 \leq i \leq n\); and \(\sum_{1 \leq i \leq n} p_{i,j} = p\).

### 3.2 Query Rewriting

We adapt the classical query rewriting algorithm PerfectRef [Calvanese et al., 2007] to decide whether \((\mathcal{P}, \mathcal{D}) \models q'\), for \(q' \in \text{Tr}(q, p)\), where \((\mathcal{P}, \mathcal{D})\) and \((q, p)\) are as in Theorem 1. When possible, we use the definitions and terminology from [Calvanese et al., 2007, Sec. 5.1], adapting some of them to our setting if needed.

For simplicity, for each role \(R^\prime\) occurring in an OBDA instance \(((\mathcal{O}, \mathcal{M}, \mathcal{S}), \mathcal{D})\), we add to \(\mathcal{O}\) the annotated role inclusions \((R^\prime \sqsubseteq R, p_{R^\prime})\) and \((R \sqsubseteq R^\prime, p'_{R^\prime})\), where \(\mathcal{T}\) is a fresh role name and \(p_{R^\prime}, p'_{R^\prime}\) are fresh variables of a provenance semiring. We assume w.l.o.g. that inverse roles only occur in such role inclusions by replacing other occurrences of \(R^\prime\) with \(\mathcal{T}\). The symbol “\(-\)” denotes non-distinguished non-shared variables. A positive inclusion \(I\) is a provenance annotated role or concept inclusion without negations. \(I\) is applicable to \(A(x, p)\) if \(I\) is annotated with \(v\) occurring in \(p\) and it has \(A\) in its right-hand side. A positive inclusion \(I\) is applicable to \(R(x, y, p)\) if \(I\) is annotated with \(v\) occurring in \(p\) and the right-hand side of \(I\) is \(R\), or \(R^\prime\) is a role inclusion annotated with \(v\) occurring in \(p\) and its right-hand side is \(R\) or \(R^\prime\). Given \(p \in \mathbb{N}_M\) and \(v \in \mathbb{N}_V\) occurring in \(p\), we denote by \(p_{v}\) the result of removing one occurrence of \(v\) from \(p\).

**Definition 1.** Let \(g\) be an atom and \(I\) a positive inclusion applicable to \(g\). The atom obtained from \(g\) by applying \(I\), denoted by \(gr(g, I)\), is defined as follows:

- \(gr(A(x, p), (A_1 \sqsubseteq A, v)) = A_1(x, p_v)\);
- \(gr(A(x, p), (\exists R \sqsubseteq A, v)) = R(x, p_v)\);
- \(gr(R(x, y, p), (A \sqsubseteq \exists R, v)) = A(x, p_v)\);
- \(gr(R(x, y, p), (R \sqsubseteq \exists R, v)) = R_1(x, y, p_v)\);
- \(gr(R(x, y, p), (R_1 \sqsubseteq R, v)) = R_1(x, y, p_v)\);
- \(gr(g, I) = R_1(y, x, p_{v})\), if \(g = R(x, y, p)\) and either \(I = (R_1 \sqsubseteq R, v)\) or \(I = (R_1 \sqsubseteq R, v)\).

**Example 4.** Consider the query

\[q = \exists xyzw.(\text{headGov}(x, y, z) \land \text{City}(y, w))\]

and the polynomial \(p = (s \times t) + (s \times r)\). Then,

\[\exists x_1 y_1 x_2 y_2.(\text{headGov}(x_1, y_1, z) \land \text{headGov}(x_2, y_2, s) \land \text{City}(y_1, t) \land \text{City}(y_2, r))\]

is in \(\text{Tr}(q, p)\).

**Theorem 1.** Let \((\mathcal{P}, \mathcal{D})\) be an OBDA instance, \(q\) a BCQ and \(p \in \mathbb{N}_p\) a polynomial formed of mathematically distinct monomials. \((\mathcal{P}, \mathcal{D}) \models (q, p)\) if \((\mathcal{P}, \mathcal{D}) \models \text{Tr}(q, p)\).

Without assuming that \(p \in \mathbb{N}_p\) is formed of mathematically distinct monomials, we would need to add inequalities to the queries in \(\text{Tr}(q, p)\) (there is no way to distinguish \(\text{Tr}(q, p) + p\) from \(\text{Tr}(q, p)\)). By Lemma 1, given the OBDA instance \((\mathcal{P}, \mathcal{D})\) and query \((q, p)\), there are \((\mathcal{P}_m, \mathcal{D}_m)\) and \((q_m, p_m)\), satisfying the assumption of Theorem 1, which we can use to decide whether \((\mathcal{P}, \mathcal{D}) \models (q, p)\). This is crucial for query entailment since entailment of conjunctive queries with inequalities in DL-Lite\(\mathcal{X}\) is undecidable [Gutiérrez-Basulto et al., 2015].
Algorithm 1 PerfectRef

**Input:** a BCQ \( q \), a set of positive inclusions \( \mathcal{O}_\mathcal{T} \)

**Output:** a set of BCQs \( \mathcal{P} \mathcal{R} \)

1: \( \mathcal{P} \mathcal{R} := \{ q \} \)
2: repeat
3: \( \mathcal{P} \mathcal{R} := \mathcal{P} \mathcal{R} \)
4: for all \( q \in \mathcal{P} \mathcal{R} \), all \( g_1, g_2 \in q \) and all \( I \in \mathcal{O}_\mathcal{T} \) do
5: if \( \{ q | g / g( I ) \} \notin \mathcal{P} \mathcal{R} \) and \( I \in \mathcal{O}_\mathcal{T} \) is applicable to \( g \in q \) then
6: \( \mathcal{P} \mathcal{R} := \mathcal{P} \mathcal{R} \cup \{ q | g / g( I ) \} \)
7: if there are \( g_1, g_2 \in q \) such that \( g_1 \) and \( g_2 \) unify then
8: \( \mathcal{P} \mathcal{R} := \mathcal{P} \mathcal{R} \cup \{ \tau( \text{reduce}(q, g_1, g_2)) \} \)
9: until \( \mathcal{P} \mathcal{R} = \mathcal{P} \mathcal{R} \)
10: return \( \mathcal{P} \mathcal{R} \)

We use PerfectRef (Algorithm 1) originally presented in [Calvanese et al., 2007], except that the applicability of a positive inclusion \( I \) to an atom \( g \) is as previously described and \( g( I ) \) follows Definition 1. Let \( q | g / g' \) denote the BCQ obtained from \( q \) by replacing the atom \( g \) with a new atom \( g' \); let \( \tau \) be a function that takes as input a BCQ \( q \) and returns a new BCQ obtained by replacing each occurrence of an unbound variable in \( q \) with the symbol \( \cdot \); and let reduce be a function that takes as input a BCQ \( q \) and two atoms \( g_1, g_2 \) and returns the result of applying to \( q \) the most general unifier of \( g_1 \) and \( g_2 \) (unifying mathematically equal terms). PerfectRef\((q, \mathcal{O}_\mathcal{T})\) is the output of the algorithm PerfectRef over \( q \) (with a monomial in \( N_M \) in the last parameter of each atom) and a set \( \mathcal{O}_\mathcal{T} \) of positive inclusions of an OBDA instance \((\mathcal{O}, \mathcal{M}, S), \mathcal{D}\).

**Example 5.** Consider an OBDA instance \((\mathcal{O}, \mathcal{M}, S), \mathcal{D}\) as in Ex. 1. We call Algorithm 1 with \( \mathcal{O}_\mathcal{T} \) and the query \( q = \exists x. \text{Mayor}(x, p \times n \times s) \) as input. Since \( I \) is applicable to \( g = \text{Mayor}(x, p \times n \times s) \), in Line 6, Alg. 1 adds to \( \mathcal{P} \mathcal{R} \) the result of replacing \( g \) by \( \text{addGov}(x, p \times n) \) in \( q \). Hence, \( q^+ = \exists x, y. \text{headGov}(x, y, p \times n) \in \text{PerfectRef}(q, \mathcal{O}_\mathcal{T}) \). Indeed \( q^+ \) is a rewriting of \( q \).

4 Computing the Provenance of a Query

We now consider the problem of computing the provenance of a query. To avoid the case of an infinite provenance, we focus on the special case where the provenance semiring is fully idempotent, which is a sufficient condition for finite provenance (Proposition 1). The semiring is fully idempotent if for every polynomial \( p \in N_\mathcal{P}, p \times p = p \) and \( p + p = p \). This is the case, e.g., if the provenance refers to the name of the source of the knowledge; having several times the same name does not affect the result. Alternatively, one can model access rights and observe whether certain pieces of knowledge are needed for the entailment of a query w.r.t. an OBDA instance.

For fully idempotent semirings, the task corresponds to computing relevant monomials. More precisely, in this special case we want to compute all monomials \( p \) such that \( (\mathcal{P}, \mathcal{D}) \models (q, p) \). The provenance of the query w.r.t. the OBDA instance is the addition of all these monomials. This definition is equivalent to the general one since the semiring is idempotent: repetitions of a monomial do not affect the result, and repetitions of a variable within a monomial can be removed. If the semiring is only multiplicatively idempotent, then computing monomials does not suffice, as some of them may appear several times. However, the problem is still simplified to find the (finite) number of repeated monomials to be observed. In general, the query polynomial may be composed of exponentially many monomials, even if the query is a simple one of the form \( \exists x.A(a, x) \), with \( A \in N_C \).

**Proposition 2.** There exists an OBDA instance \((\mathcal{P}, \mathcal{D})\) and a simple query \( q \) such that the provenance polynomial of \( q \) w.r.t. \((\mathcal{P}, \mathcal{D})\) is formed of exponentially many monomials.

For some queries, provenance cannot be expressed by a provenance polynomial of polynomial length in the size of the ontology, even if an expanded form is not required. This follows from known results in monotone complexity [Karchmer and Wigderson, 1990]: there is no monotone Boolean formula (i.e., propositional formula using only the connectives \( \land \) and \( \lor \)) of polynomial length expressing all the simple paths between two nodes in a graph. This holds already for complete graphs. Graphs can be described in DL-Lite\( _\mathcal{F} \) (and simpler logics) using basic inclusion axioms, and monotone Boolean formulas are provenance polynomials over an idempotent semiring, where the \( \land \) and \( \lor \) serve as product and addition. Hence we have the following result [Peñaloza, 2009].

**Proposition 3.** There exist an OBDA instance \((\mathcal{P}, \mathcal{D})\) and a query \( q \) such that the provenance of \( q \) w.r.t. \((\mathcal{P}, \mathcal{D})\) cannot be represented in polynomial space. This holds even for idempotent semirings, and if every axiom has a unique label.

On the other hand, if every axiom is labeled with a unique variable, then the provenance polynomial for instance queries can be computed efficiently, whenever its length does not increase greatly; that is, it can be computed in polynomial time in the size of the input and the output. The proof of this claim follows the same ideas from [Peñaloza and Sertkaya, 2017], based on the fact that all the relevant monomials from the provenance are enumerable with polynomial delay.

**Lemma 2.** The provenance \( p \) of an instance query w.r.t. an OBDA instance \((\mathcal{P}, \mathcal{D})\) can be computed in polynomial time.
Algorithm 2 ComputeProv

Input: a BCQ $q_0$, an OBDA instance $((\mathcal{O}, \mathcal{M}, S), D)$
Output: the provenance $p$ of $q$ w.r.t. $((\mathcal{O}, \mathcal{M}, S), D)$

1: $PR := \text{PerfectRef}^+ (q_0, T)$,
2: for all $q \in PR$ do
3:   for all matches $\pi$ of $q_\pi$ in $\mathcal{I}_M (D)$ do
4:     $PR := PR \cup \{ q_\pi^* \}$
5: $PR := PR \setminus \{ q \}$
6: return $p := \sum_{q \in PR} \prod_{P(i,t) \in q} t$

in the size of $(P, D)$ and of the polynomial $p$.

We give an algorithm for computing the provenance of a BCQ w.r.t. an OBDA instance. We focus on BCQs that do not have monomials in the last term of the atom. A BCQ $q = \exists \bar{x} . \varphi (\bar{x}, \bar{a})$ is standard if, for all $P(t, t) \in q$, $t$ is a fresh variable in $\bar{x}$. Algorithm 2 computes the provenance of a standard BCQ w.r.t. an OBDA instance. We adopt the same notation used for describing PerfectRef [Calvanese et al., 2007] (also used in Section 3). PerfectRef* is a variant of PerfectRef (Algorithm 1), where the notions of applicability of an inclusion $I$ w.r.t. an atom $g$ and the definition of $gr(g, I)$ are as follows. $I$ is applicable to an atom $A(x, p)$ if $I$ has $A$ in its right-hand side. A positive inclusion $I$ is applicable to an atom $R(x, y, p)$ if (i) $x = \_.$ and the right-hand side of $I$ is $\exists R$, or (ii) the right-hand side of $I$ is either $R$ or $R^-$. Given $p \in N_M$ and $v \in N_V$, we define $p^v$ as $p \times v$ if $v$ does not occur in $p$, and we define $p^v$ as $p$, otherwise. E.g., $vw^v = vw$.

Definition 2. Let $g$ be an atom and $I$ a positive inclusion applicable to $g$. The atom obtained from $g$ by applying $I$, denoted by $gr(g, I)$, is defined as follows:

- $gr(A(x, p), (A_1 \sqsubset A, v)) = A_1(x, p^v)$;
- $gr(A(x, p), (\exists R \sqsubset A, v)) = R(x, \_., p^v)$;
- $gr(R(x, \_., p), (A \sqsubset \exists R, v)) = A(x, p^v)$;
- $gr(R(x, \_., p), (\exists R \sqsubset R, v)) = R(x, \_., p^v)$;
- $gr(R(x, y, p), (R_1 \sqsubset R, v)) = R_1(x, y, p^v)$;
- $gr(g, I) = R_1(y, x, p^v)$, if $g = R(x, y, p)$ and either $I = (R \sqsubset R^-)$ or $I = (R_1 \sqsubset R)$. $\diamond$

For standard BCQs, Algorithm 2 is sound and complete. Termination of Algorithm 2 is an easy consequence of termination of PerfectRef. The main difference between Algorithm 2 and Algorithm 1 (Section 3) is that here we assume that a standard BCQ is given (without any provenance information) and we aim at computing its provenance. Instead of removing variables of the semiring while applying positive inclusions (Definition 1), we add the variables of the semiring whenever the associated positive inclusion is applied (Definition 2). In Line 1, we write $q^*$ to denote the result of replacing each $t$ in $P(\bar{t}, t) \in q$ by $\_.$, where $\_.$ is a fresh symbol from $N_V$. This transformation ensures that in Definition 2 the last term is always an element of $N_M$. In Line 3, we denote by $q_\pi$ the result of replacing, for each $P(\bar{t}, t) \in q$, the last term $t$ by a fresh variable from $\tilde{y}$ (i.e., $q_\pi$ is a standard BCQ). We perform another transformation in Line 4, denoted by $q_\pi^* \pi$, which is the result of replacing, for each $P(\bar{t}, t) \in q$, the symbol $\_.$ in $t$ by $u \in N_M$ such that $u^\pi = \pi (y)$ (if there are multiple mathematically equal such $u$, we simply choose $u$ arbitrarily), where $y$ is the last term of the corresponding atom in $q_\pi$ (that is, $P(\bar{t}, y) \in q_\pi$). Observe that $\pi$ is a match of $q_\pi$ in $\mathcal{I}_M (D)$. Example 6. Assume Algorithm 2 computes as input the standard query $q_0 = \exists xz. \text{Mayor}(x, z)$ and an OBDA instance $((\mathcal{O}, \mathcal{M}, S), D)$ with $O = \{ \text{headGov} \sqsubseteq \text{Mayor} \}$ and $\mathcal{I}_M (D) = \{ \text{headGov}(\text{Renier}, \text{Venice}), u \}, \{ \text{headGov} (\text{Brugnaro}, \text{Venice}), v \}$.

In Line 1, Algorithm 2 calls PerfectRef*, defined as a variant of PerfectRef (Algorithm 1), where the notions of applicability of an inclusion $I$ w.r.t. an atom $g$ and the definition of $gr(g, I)$ are as in Section 4. The return of PerfectRef* is $PR = \{ \exists x. \text{Mayor}(x, \_), \exists xz. \text{headGov}(x, z, \_ \times s) \}$. Then, for all $q \in PR$ and all matches $\pi$ of $q_\pi$ in $\mathcal{I}_M (D)$ (if they exist) the algorithm adds $q_\pi^* \pi$ to $PR$. In this example, assume $q = \exists xz. \text{headGov}(x, z, \_ \times s)$. We have two matches of $q_\pi = \exists xz y. \text{headGov}(x, z, y)$ in $\mathcal{I}_M (D)$, one mapping $y$ to $u$ (call this match $\pi$) and the other mapping $y$ to $v$ (call it $\pi'$). So, $q_\pi^* \pi = \exists xz. \text{headGov}(x, z, u \times s)$ and $q_\pi^* \pi' = \exists xz. \text{headGov}(x, z, v \times s)$ in Line 5, Algorithm 2 removes $q_\pi^* \pi$ from $PR$. Finally, in Line 6, it returns the polynomial $u \times s + v \times s$. $\diamond$

Theorem 4. Let $q$ be a standard BCQ and $(P, D)$ an OBDA instance. Given $q$ and $(P, D)$ as input to Algorithm 2, it outputs the provenance of $q$ w.r.t. $(P, D)$.

The upper bounds from the previous section for the general case obviously apply in the restricted idempotent case as well.

5 Evaluation

To evaluate the feasibility of our approach, we implemented a prototype system (OntoProv) that extends the state-of-the-art OBDA system Ontop [Calvanese et al., 2017] with the support for provenance. Ontop supports SPARQL query answering over ontologies in OWL 2 QL, the W3C standard corresponding to DL-Lite$^\mathcal{R}$ [Motik et al., 2012]. The algorithm of Ontop has two stages, an offline stage, which classifies the ontology and saturates the input set of mappings, and an online stage, which rewrites the input queries according to the saturated set of mappings. OntoProv enriches these steps by taking into account provenance information, and relies on ProvSQL [Senellart et al., 2018] to handle provenance from the database and queries in the mappings that go beyond the CQ fragment. We compare OntoProv v3.0.0-beta-3 and OntoProv over the BSBM [Bizer and Schultz, 2009] and the NPD [Lanti et al., 2015] benchmarks. Experiments were run on a server with 2 Intel Xeon X5690 Processors (24 logical cores at 3.47 GHz), 106 GB of RAM and five 1 TB 15K RPM HDs. As RDBMS we have used PostgreSQL 11.2.

5.1 Evaluation with the BSBM Benchmark

The BSBM benchmark is designed to test the different features of SPARQL. It provides a baseline for our tests, since it comes with an empty ontology and therefore it does not require ontological reasoning. In this experiment we restrict to a set of parametric queries (called here query mix) in the benchmark
Table 2: NPD Experiment (times in seconds)

<table>
<thead>
<tr>
<th></th>
<th>mixTime</th>
<th></th>
<th>mixTime</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bsbm10k</td>
<td>2.0s</td>
<td></td>
<td>3.2s</td>
<td></td>
</tr>
<tr>
<td>bsbm1M</td>
<td>326s</td>
<td></td>
<td>364s</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: BSBM Experiment

<table>
<thead>
<tr>
<th>Q #unf t t10</th>
<th>#unf #uinst</th>
<th>Ontop OntoProv</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 29.5 172.4</td>
<td>16 16</td>
<td>5.2 1.7 1524 49</td>
</tr>
<tr>
<td>2 0 5 31</td>
<td>32 32</td>
<td>3 4 28 60</td>
</tr>
<tr>
<td>3 24 5.0 51.1</td>
<td>16 16</td>
<td>248.0 296.5 84 153</td>
</tr>
<tr>
<td>4 0 3.2 24.3</td>
<td>16 16</td>
<td>437.6 297.3 90 53</td>
</tr>
<tr>
<td>5 0 1 2</td>
<td>0 0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>6 13 107.5 804.3</td>
<td>369 369</td>
<td>1439.3 426 tout</td>
</tr>
<tr>
<td>7 0 4 2</td>
<td>0 0</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>9 15 6.8 53.4</td>
<td>64 55</td>
<td>9 5 5</td>
</tr>
<tr>
<td>10 1 4 25.4</td>
<td>4 4</td>
<td>8 11.3 1 7</td>
</tr>
<tr>
<td>11 6 53.6 760.8</td>
<td>184 184</td>
<td>1342.6 474 tout</td>
</tr>
<tr>
<td>12 8 69.1 1215.5</td>
<td>185 185</td>
<td>1248.1 476 tout</td>
</tr>
<tr>
<td>31 21 60.3 633.0</td>
<td>248 239</td>
<td>1.9 5.3 120 60</td>
</tr>
</tbody>
</table>

5.2 Evaluation with the NPD Benchmark

As opposed to BSBM, the NPD Benchmark is specifically tailored to OBDA systems; it comes with a complex ontology, complex mappings, and queries of various kinds. We restrict to 12 user queries that are supported by our framework. We use the dataset NPD, containing real-world data about the oil extraction domain, and the dataset NPD10, which is 10 times the size of NPD and is generated by a data scaler [Lanti et al., 2019]. Differently from the BSBM benchmark, in NPD we observed many timeouts (set to 40 minutes) when running the benchmark queries with OntoProv. This is due to the fact that, in NPD, the optimizations performed by Ontop over the query unfoldings are crucial for getting reasonably compact SQL queries. Such optimizations, however, need to be disabled in OntoProv to guarantee completeness. In fact, we are interested in all the possible ways to derive a result, and cannot identify and discard redundant derivations. For a broader discussion about these aspects, please refer to the additional material.

We assume that a user of OntoProv is more interested in understanding the reason for a specific answer tuple, rather than getting in bulk all possible explanations for all possible answer tuples. To simulate such user interaction, in our tests we have instantiated the NPD queries with answer tuples, and have run the obtained instantiated queries (which are, in fact, BCQs) over OntoProv. Table 2 contains the aggregate results of our runs. For each of our tests, we performed 5 test runs.

The columns #unf and #uinst denote the number of times a UNION operator appears in the unfolding of an NPD query and an instantiated query, respectively. This measure gives an idea on the complexity of the unfolding, and we can observe that the unfoldings produced by OntoProv are much more complex than those produced by Ontop. As argued above, this is because OntoProv disallows some optimizations. Columns t and t10 denote the average execution times of the queries over the datasets NPD and NPD10, respectively, and for instantiated queries these values are respectively denoted by tinst and tinst10. The execution times for OntoProv are generally much higher than for Ontop. We attribute this to the increased complexity of the unfoldings. Columns #prov and #prov10 denote the number of results for the instantiated queries, respectively over NPD and NPD10. These numbers can be interpreted as the number of possible ways an answer tuple can be derived, and give an indication on the complexity of the benchmark itself. For instance, for query 1 over the NPD dataset there are on average 1524 explanations for a single answer tuple.

This test shows that the approach is feasible even with complex ontologies and mappings, but also that more work is needed in order to devise optimization techniques dedicated to a setting with provenance.

6 Conclusions and Discussion

We investigated the problem of dealing with provenance within OBDA, based on the provenance semiring approach introduced for databases. In our case, every element of an OBDA instance is annotated with provenance information. We showed that query rewriting techniques can be applied to deal with provenance as well. An evaluation based on a prototypical implementation shows that our methods are feasible in practice.

A key difference between the problem of provenance computation (or its decision version) and that of axiom pinpointing [?; ?; ?] and query explanation [?; ?; ?] is that axiom pinpointing and query explanation focus on tracing the minimal causes of a consequence (or the lack of it). In contrast, all possible derivations are relevant for provenance, independently of whether a cause is minimal or not.

As future work, we plan to investigate provenance with the monus operator. We will also study the provenance of SPARQL query answering [?] in OBDA. Our implementation computes the provenance of a query assuming that the semiring is multiplicatively idempotent. While this assumption is useful to identify which parts of the knowledge base contribute to the query result, it restricts the applicability of our approach to other settings, in particular, to the numerical ones. For capturing probabilities, it is important to distinguish repetitions, so (multiplicative) idempotency is not suitable. In our setting, dropping the idempotency condition leads to cases where the polynomial can be infinite. It would be interesting to investigate whether the polynomial can be finitely represented, so that its computation could be applied in a numerical setting.

Acknowledgements

This research has been supported by the “European Region Tyrol-South Tyrol-Trentino” (Euregio IPN12 KAOS) and by the Free University of Bozen-Bolzano through the projects STyLoLa, OBATS, QUADRO, PROVDL, and FO2S.
References


